网络空间安全创新创业实践

The Rho method attack

实验报告

王祥宇---202000460053

项目时间：2022 年 7 月 10 日

目录

一、 项目任务 3

二、 实验过程及思路 3

三、 实验结果及检验 5

四、 实验反思与总结 6

五、 参考文献 6

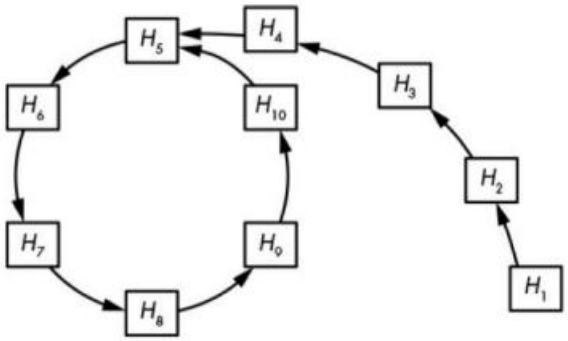
一、项目任务

Project: implement the Rho method of reduced SM3

此项目中用到的 SM3 算法在生日攻击项目中已经给出思路解释，在此不再赘述。

二、实验过程及思路

说到 Rho 方法我想到的就是已经学过的因子分解方法之一 Pollard Rho 算法，而且该项目的 Rho 算法和 Pollard Rho 算法十分类似，具体思路都是如下图：



即从一个随便的消息 m 出发，不断 hash 消息 m 的 hash 值这样直到找到一个圈，构成图中的 Rho 形状，这样就找到了碰撞。所以需要首先实现 repeat(x) 函数用来对消息 x 的 hash 值 hash 一次，即相当于图中从 Hi 到 H(i+1)，具体实现如下：

```
def repeat(x):#返回sm3hash(x)
    if isinstance(x,int):
        return sm3hash(hex(x))
    temp=int(str(x),16)
    return sm3hash(hex(temp))
```

The Rho method attack

在 Rho 算法中可以发现图像中的第一个不是消息，而是消息的 hash 值。所以为了后面实现的方便，在 repeat 函数中首先要判断 x 是否为 int 型，如果是，说明 x 就是消息 m 而不是他的 hash 值，所以直接返回 m 的 hash 即可。如果不是 int 型，说明传入的 x 是 Hi，所以需要将他先转化为 int 然后再调用 hash 返回。

有了上面的 repeat 函数，就可以实现 Rho_method(x)函数了，用于寻找前 x 比特的碰撞。首先和生日攻击一样，由于 hash 值为十六进制表示，所以这里规定 x 为 4 的倍数。然后第一步先要构造 Rho 圈。在 Pollard Rho 算法中，是分两条线路：H1-H2-H3-...和 H2-H4-H6-...，然后一次比较两条线路直到出现碰撞。在此采用和 Pollard Rho 算法一样的思路。即随机选取一个消息 m，然后对其调用一次 repeat 得到 H1，在对 H1 调用一次 repeat 得到 H2，这样就得到了两条线路的起点。（存在和生日攻击同样的问题，由于比较的是十六进制数，所以可能存在多 1 到 3bit 的碰撞是无法检测出来的）

```
x_a=sm3hash(hex(n))
#temp=int(str(x_a),16)
x_b=repeat(x_a)
```

接下来就是 while 循环找碰撞即可，即构造 Rho 圈：

```
while x_a[:num]!=x_b[:num]:
    x_a=repeat(x_a)
    x_b=repeat(repeat(x_b))
    p+=1
```

其中的 p 即为 Rho 圈的大小。若找到，则 while 结束，开始从 1-p 进行遍历找到碰撞输出即可：

```
for i in range(p):
    if repeat(x_a)[:num]==repeat(x_b)[:num]:
        return int(str(x_a),16),int(str(x_b),16)
    else:
        x_a=repeat(x_a)
        x_b=repeat(x_b)
```

完整实现代码见 python 文件。

三、实验结果及检验

当 x=12，结果如下：

```
=====
1643
(31567245717694923371196878058996019188206500339240998987516367321121940145401,
10604323839093741528460788614528788054455378670105575782403229024718257180238O)
```

对其进行检验：

```
#print(Rho_method(20))
print(sm3hash(hex(31567245717694923371196878058996019188206500339240998987516367321121940145401)))
print(sm3hash(hex(10604323839093741528460788614528788054455378670105575782403229024718257180238O)))
```

结果如下：

```
ce20534ebf1ed17c5949ae01f02e225d487e6e0d64ac85360dea1081f7e3c85e
ce23cb06c2ca1a88dc872948bca57ab81113c42c0b96f36e0cb94760a89c3462
```

可以发现前三位均是'ce2'，但是由于'0'和'3'的前两个比特是相同的，所以 x12 时一共找到 14bit 的碰撞。

当 x=16 时，结果如下：

```
_____
11910
(92149844787472072187967210329932186750909454787526956128423713225310314973480,
11481571481194647871726007103240976528591948648858242850194153842273103801473O)
```

对其进行检验：

```
print(sm3hash(hex(92149844787472072187967210329932186750909454787526956128423713225310314973480)))
print(sm3hash(hex(11481571481194647871726007103240976528591948648858242850194153842273103801473O)))
```

结果如下：

```
_____
a52bfa59ce83396cd393b75a28e6fcddadc067835259351d98ec440ec578d6df
a52b810c548cbd6f58e451b481559bddf2ed56f6ff391d8baa6fbac080864bbd
```

可以发现前四位均是'a52b'，所以一共找到 16bit 的碰撞。

四、实验反思与总结

本次项目参考了 Pollard Rho 算法，所以实现起来思路比较清晰。通过本次项目了解了寻找 SM3 碰撞的 Rho 算法，为以后的密码学习奠定了基础。比较遗憾的是就是 Rho 算法找到的碰撞比生日攻击要短，可能是因为 Rho 算法消息都比较大的关系叭。而且当 x=20 时已经跑不出来了，连 p 都没有输出，说明 x=20 时连 Rho 圈都没有构造出来。

五、参考文献

【SM3 算法中的每一步操作均可以在下面的文献中找到】

http://www.sca.gov.cn/sca/xwdt/2010-12/17/1002389/files/302a3ada057c4a73830536d03e683110.pdf